ARCHModels.jl: Estimating ARCH models in Julia

Simon A. Broda*, Marc S. Paolella†

Department of Banking and Finance, University of Zurich

March 2020

Abstract

This paper introduces **ARCHModels.jl**, a package for the Julia programming language that implements a number of univariate and multivariate ARCH-type models. This model class is the workhorse tool for modelling the conditional volatility of financial assets. Their distinguishing feature is that they model the latent volatility as a (deterministic) function of past returns and volatilities. This recursive structure results in loop-heavy code which, due to its just-in-time compiler, Julia is well-equipped to handle. As such, the entire package is written in Julia, without any binary dependencies. We benchmark the performance of **ARCHModels.jl** against popular implementations in MATLAB, R, and Python, and illustrate its use in a detailed case study.

Key Words: ARCH, GARCH, CCC, DCC, Value at Risk, Julia.

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 750559).

* *E-mail address:* simon.broda@bf.uzh.ch.

† *E-mail address:* marc.paolella@bf.uzh.ch

Electronic copy available at: https://ssrn.com/abstract=3551503

1 Introduction

Financial returns data at daily or higher frequency display a number of *stylized facts*, including volatility clustering (large (in absolute value) returns tend to cluster together), heavy tails, and statistical leverage, among others; see Figure 1 for an example. Modeling these lies at the heart of much of financial econometrics, because the volatility and conditional distribution of an asset (or a group of assets) are key ingredients in applications such as risk management, portfolio optimization, and derivative pricing. ARCH models (autoregressive conditional heteroskedasticity) form the most widely used class of models for capturing these features. In an ARCH-type model, the latent volatility σ_t^2 of an asset is modeled in terms of past returns and volatilities. For example, the basic GARCH(1,1) model for a sample of daily asset returns $\{r_t\}_{t\in\{1,\dots,T\}}$, due to Bollerslev (1986), is

$$r_t = \sigma_t z_t, \quad z_t \sim \mathrm{N}(0,1), \quad \sigma_t^2 = \omega + \alpha r_{t-1}^2 + \beta\sigma_{t-1}^2, \quad \omega,\alpha,\beta > 0, \quad \alpha+\beta < 1. \tag{1.1}$$

The GARCH model extends the ARCH model (in which $\beta = 0$) of Engle (1982), who in 2003 was awarded a Nobel Memorial Prize in Economic Sciences for its development.

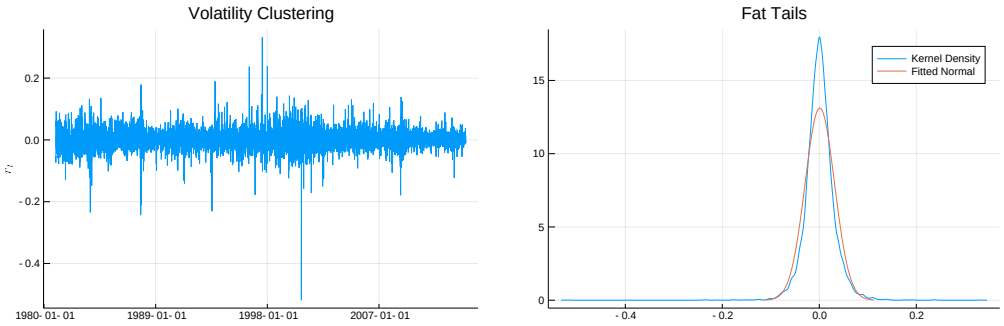

Figure 1: Volatility clustering (left panel) and heavy tails (right panel) illustrated by way of daily returns on Apple stock

Due to their popularity, the ARCH model and its various extensions have been implemented in many commercial and free programming environments; examples include the **rugarch** (Ghalanos, 2019) package for R (R Core Team, 2019), the **ARCH** (Sheppard and other contributors, 2019) package for Python (van Rossum, 1995), and MATLAB's (The MathWorks Inc., 2019) **Econometrics** toolbox. These implementations all outsource the evaluation of the likelihood function to a compiled language, because the recursive nature of (1.1) defies any attempt at "vectorizing" it, a common recommendation for performant implementations in interpreted languages. Julia (Bezanson et al., 2017) is unique among these high-level languages because its just-in-time compiler allows us to keep even the tight loops in Julia itself without sacrificing performance; in fact, our implementation outperforms those mentioned above; see Section 5.2. This is partly due to the use of automatic differentiation for the computation of gradients in maximizing the likelihood, via **Optim.jl** (Mogensen and Riseth, 2018) and **ForwardDiff.jl** (Revels et al., 2016).

ARCHModels.jl is a registered Julia package and is easily installed with Julia's package manager using the commands `using Pkg; Pkg.add("ARCHModels")`. It implements estimation, model selection, simulation, and Value at Risk (VaR) calculations for a variety of univariate (ARCH, GARCH, TGARCH, EGARCH) and multivariate (CCC, DCC) ARCH models, for different choices of standardized innovation distributions (Gaussian, Student's t, Generalized error distribution). The conditional mean can be specified as either zero, an intercept, a linear regression model, or an ARMA(p, q) model. The package is designed to be easy to extend with other volatility specifications and distributions, and integrates with the relevant parts of the Julia ecosystem, such as **DataFrames.jl** (Harris et al., 2015), **Distributions.jl** (Lin et al., 2019b), **GLM.jl** (Bates and other contributors, 2012), **HypothesisTests.jl** (Kornblith and other contributors, 2012), and **StatsBase.jl** (Lin et al., 2019a).

The remainder of the paper is as follows: Section 2 defines the various models implemented in **ARCHModels.jl**. Section 3 describes the design of the package, in particular the type hierarchy. Section 4 presents a case study detailing the use of the package. Section 5 offers a comparison with implementations in other languages. Section 6 concludes.

Electronic copy available at: https://ssrn.com/abstract=3551503

2 ARCH models

Consider a sample of daily asset returns $\{r_t\}_{t\in\{1,\dots,T\}}$. All models covered in this package share the same basic structure, in that they decompose the return into a conditional mean and a mean-zero innovation. In the univariate case,

$$r_t = \mu_t + a_t, \quad \mu_t \equiv \mathbb{E}[r_t \mid \mathcal{F}_{t-1}], \quad \sigma_t^2 \equiv \mathbb{E}[a_t^2 \mid \mathcal{F}_{t-1}],$$

$z_t \equiv a_t/\sigma_t$ is identically and independently distributed according to some law with mean zero and unit variance, and $\{\mathcal{F}_t\}$ is the natural filtration of $\{r_t\}$ (i.e., it encodes information about past returns). In the multivariate case, $r_t \in \mathbb{R}^d$, and the general model structure is

$$r_t = \mu_t + a_t, \quad \mu_t \equiv \mathbb{E}[r_t \mid \mathcal{F}_{t-1}], \quad \Sigma_t \equiv \mathbb{E}[a_t a_t^{\mathrm{T}}] \mid \mathcal{F}_{t-1}].$$

ARCH models specify the conditional volatility σ_t (or in the multivariate case, the conditional covariance matrix Σ_t) in terms of past returns, conditional (co)variances, and potentially other variables.

3 ARCHModels.jl: ARCH models in Julia

3.1 Univariate type hierarchy

This package represents an ARCH model as an instance of either `UnivariateARCHModel` or `MultivariateARCHModel`. These are subtypes of `ARCHModel` and implement the interface of `StatisticalModel` from **StatsBase**.

An instance of `UnivariateARCHModel` contains a vector of data (such as equity returns), and encapsulates information about the volatility specification (e.g., GARCH or EGARCH), the mean specification (e.g., whether an intercept is included), and the error distribution.

3.1.1 Volatility specifications

Volatility specifications describe the evolution of σ_t. They are modelled as subtypes of `UnivariateVolatilitySpec`. There is one type for each class of (G)ARCH model, parameterized by the number(s) of lags (e.g., p, q for a GARCH(p, q) model).

The simplest volatility specification is given by the ARCH(q) model, due to Engle (1982). It reads

$$\sigma_t^2 = \omega + \sum_{i=1}^{q} \alpha_i a_{t-i}^2, \quad \omega, \alpha_i > 0, \quad \sum_{i=1}^{q} \alpha_i < 1. \tag{3.1}$$

The corresponding type is `ARCH{q}`.

The GARCH(p, q) model, due to Bollerslev (1986), generalizes the ARCH(q) model by including lagged values of the squared volatility on the right hand side of (3.1). This renders the conditional variance as

$$\sigma_t^2 = \omega + \sum_{i=1}^{p} \beta_i \sigma_{t-i}^2 + \sum_{i=1}^{q} \alpha_i a_{t-i}^2, \quad \omega, \alpha_i, \beta_i > 0, \quad \sum_{i=1}^{\max p,q} \alpha_i + \beta_i < 1.$$

It is available as `GARCH{p, q}`.

The ARCH and GARCH models are special cases of a more general class of models, known as TGARCH (Threshold GARCH), due to Glosten et al. (1993). The TGARCH(o, p, q) model takes the form

$$\sigma_t^2 = \omega + \sum_{i=1}^{o} \gamma_i a_{t-i}^2 1_{a_{t-i}<0} + \sum_{i=1}^{p} \beta_i \sigma_{t-i}^2 + \sum_{i=1}^{q} \alpha_i a_{t-i}^2, \quad \omega, \alpha_i, \beta_i, \gamma_i > 0, \quad \sum_{i=1}^{\max o,p,q} \alpha_i + \beta_i + \gamma_i/2 < 1.$$

The TGARCH model allows the volatility to react differently (typically more strongly) to negative shocks, a feature known as the (statistical) leverage effect. Is available as `TGARCH{o, p, q}`.

Finally, The EGARCH(o, p, q) volatility specification, due to Nelson (1991), is

$$\log(\sigma_t^2) = \omega + \sum_{i=1}^{o} \gamma_i z_{t-i} + \sum_{i=1}^{p} \beta_i \log(\sigma_{t-i}^2) + \sum_{i=1}^{q} \alpha_i (|z_{t-i}| - \sqrt{2/\pi}), \quad z_t = r_t/\sigma_t, \quad \sum_{i=1}^{p} \beta_i < 1.$$

Electronic copy available at: https://ssrn.com/abstract=3551503

Like the TGARCH model, it can account for the leverage effect. The corresponding type is `EGARCH{o, p, q}`.

The constructors for the volatility specifications take as input a coefficient vector, where the order of the parameters is such that all parameters pertaining to the first type parameter (p) appear before those pertaining to the second (q). For example, an EGARCH(1, 1, 1) model with $\omega = -0.003$, $\gamma_1 = -0.03$, $\beta_1 = 0.99$ and $\alpha_2 = .2$ is obtained with

```
julia> using ARCHModels

julia> EGARCH{1, 1, 1}([-0.003, -0.03, 0.99, 0.2]);
```

Explicitly creating instances of volatility specifications is only necessary for simulation (see Section 4); for fitting, passing the type is sufficient.

3.1.2 Mean specifications

Mean specifications serve to specify μ_t. They are modelled as subtypes of `MeanSpec`. They contain their parameters as (possibly empty) vectors. Convenience constructors are provided where appropriate, though as with volatility specifications, constructing them explicitly is only required for simulation, not for fitting. The following specifications are available:

A zero mean, i.e., $\mu_t = 0$, is available as `NoIntercept`; An intercept ($\mu_t = \mu$) is obtained as `Intercept`. An ARMA(p, q) model, which specifies the conditional mean as

$$\mu_t = c + \sum_{i=1}^{p} \varphi_i r_{t-i} + \sum_{i=1}^{q} \theta_i a_{t-i},$$

is available as `ARMA{p, q}`. The special cases of pure AR(p) and MA(q) models are also available, as `AR{p}` and `MA{q}`, respectively.

Finally, `Regression` allows one to specify a linear regression model, as in

$$\mu_t = \mathbf{x}_t^{\mathrm{T}} \boldsymbol{\beta}.$$

Unlike the other mean specifications, a regression requires external data, which the constructor expects as a matrix with observations in rows and variables in columns, as follows:

```
julia> reg = Regression(ones(100, 1));
```

In this example, we created a regression model containing one regressor, given by a column of ones; this is equivalent to including an intercept in the model. The latter is however more memory efficient, as no design matrix needs to be stored. Another way to create a linear regression with ARCH errors is to pass a `LinearModel` or `TableRegressionModel` from **GLM.jl**, as described in Section 4.

3.1.3 Distributions

Different standardized (mean zero, variance one) distributions for z_t are available as subtypes of `StandardizedDistribution`, which in turn subtypes `Distribution{Univariate, Continuous}` from **Distributions.jl**, though not the entire interface need necessarily be implemented. Instances of `StandardizedDistribution` again hold their parameters as vectors, but convenience constructors are provided. Available distributions include the standard normal (`StdNormal`), standardized Student's t (`StdT`), and standard generalized error distribution (`StdGED`). In addition, it is possible to wrap a continuous univariate distribution from the **Distributions** package in the `Standardized` wrapper type. Below, we reimplement the standardized normal distribution:

```
julia> using Distributions

julia> const MyStdNormal = Standardized{Normal};
```


Electronic copy available at: https://ssrn.com/abstract=3551503

`MyStdNormal` can be used whereever a built-in distribution could, albeit with a speed penalty. Note also that if the underlying distribution (such as `Normal` in the example above) contains location and/or scale parameters, then these are no longer identifiable, which implies that the estimated covariance matrix of the estimators will be singular. A final remark concerns the domain of the parameters: the estimation process relies on a starting value for the parameters of the distribution, say $\theta \equiv (\theta_1, \ldots, \theta_p)^{\mathrm{T}}$. For a distribution wrapped in `Standardized`, the starting value for θ_i is taken to be a small positive value, say ε. This will fail if ε is not in the domain of θ_i; as an example, the standardized Student's t distribution is only defined for degrees of freedom larger than 2, because a finite variance is required for standardization. In that case, it is necessary to define a method of the (non-exported) function `startingvals` that returns a feasible vector of starting values, as follows:

```
julia> const MyStdT = Standardized{TDist};

julia> ARCHModels.startingvals(::Type{<:MyStdT}, data::Vector{T}) where T = T[3.]
```

3.2 Working with `UnivariateARCHModels`

The constructor for `UnivariateARCHModel` takes two mandatory arguments: an instance of a subtype of `UnivariateVolatilitySpec`, and a vector of returns. The mean specification and error distribution can be changed via the keyword arguments `meanspec` and `dist`, which respectively default to `NoIntercept` and `StdNormal`. For example, to construct a GARCH(1, 1) model with t-distributed errors, one would do

```
julia> spec = GARCH{1, 1}([1., .9, .05]);

julia> data = BG96;

julia> am = UnivariateARCHModel(spec, data; dist=StdT(3.));
```

The model can then be fitted as follows:

```
julia> fit!(am)

ARCHModels.TGARCH{0,1,1} model with Student's t errors, T=1974.


Volatility parameters:
```

	Estimate	Std.Error	z value	Pr(>|z|)
ω	0.00231698	0.00162444	1.42632	0.1538
β_1	0.884605	0.0366123	24.1614	<1e-99
α_1	0.124667	0.040124	3.10704	0.0019

```
Distribution parameters:
```

	Estimate	Std.Error	z value	Pr(>|z|)
ν	4.11924	0.401118	10.2694	<1e-24

It should, however, rarely be necessary to construct a `UnivariateARCHModel` manually via its constructor; typically, instances of it are created by calling `fit`, `selectmodel`, or `simulate`.

As discussed earlier, `UnivariateARCHModel` implements the interface of `StatisticalModel` from **StatsBase**, so one may can call `coef`, `coefnames`, `confint`, `dof`, `informationmatrix`, `isfitted`, `loglikelihood`, `nobs`, `score`, `stderror`, `vcov`, etc. on its instances. Other useful methods include `means`, `volatilities`, and `residuals`.

Electronic copy available at: https://ssrn.com/abstract=3551503

3.3 Multivariate type hierarchy

Analogously to the univariate case, an instance of `MultivariateARCHModel` contains a matrix of data (with observations in rows and assets in columns), and encapsulates information about the covariance specification (e.g., CCC or DCC), the mean specification, and the error distribution.

`MultivariateARCHModel`s support many of the same methods as `UnivariateARCHModel`s, with a few noteworthy differences: the prediction targets for `predict` are `:covariances` and `:correlations` for predicting Σ_t and R_t, respectively, and the new functions `covariances` and `correlations` respectively return the in-sample estimates of Σ_t and R_t.

3.3.1 Covariance specifications

The main challenge in multivariate ARCH modelling is the *curse of dimensionality*: allowing each of the $d(d+1)/2$ elements of Σ_t to depend on the past returns of all d other assets requires $O(d^4)$ parameters without imposing additional structure. **ARCHModels.jl** focusses on conditional correlation models, which approach this issue by decomposing Σ_t as

$$\Sigma_t = D_t R_t D_t,$$

where R_t is the conditional correlation matrix and D_t is a diagonal matrix containing the volatilities of the individual assets, which are modelled as univariate ARCH processes. The dynamics of Σ_t are modelled as subtypes of `MultivariateVolatilitySpec`. Two options are available:

DCC The dynamic conditional correlation (DCC) model of Engle (2002) imposes a GARCH-type structure on the R_t. In particular, for a DCC(p, q) model (with covariance targeting),

$$R_{ij,t} = \frac{Q_{ij,t}}{\sqrt{Q_{ii,t}Q_{jj,t}}},$$

where

$$Q_t \equiv \bar{Q}(1 - \bar{\alpha} - \bar{\beta}) + \sum_{i=1}^{p} \beta_i Q_{t-i} + \sum_{i=1}^{q} \alpha_i \epsilon_{t-i}\epsilon_{t-i}^{\mathrm{T}},$$

$\bar{\alpha} \equiv \sum_{i=1}^{q} \alpha_i$, $\bar{\beta} \equiv \sum_{i=1}^{q} \beta_i$, $\epsilon_t \equiv D_t^{-1} a_t$, $Q_t = \mathrm{cov}(\epsilon_t | F_{t-1})$, and $\bar{Q} = \mathrm{cov}(\epsilon_t)$. It is available as `DCC{p, q}`. The constructor takes as inputs $\bar{Q}$, a vector of coefficients, and a vector of `UnivariateARCHModel`s:

```
julia> DCC{1, 1}([1. .5; .5 1.], [.9, .05], [GARCH{1, 1}([1., .9, .05]) for _ in 1:2])
DCC{1, 1, ARCHModels.TGARCH{0,1,1}} specification.

                  β₁     α₁
Parameters:      0.9   0.05
```

The DCC model is typically estimated in two steps, by first fitting univariate ARCH models to the individual assets and saving the standardized residuals $\{\epsilon_t\}$, and then estimating the DCC parameters from those. Engle (2002) provides the details and expressions for the standard errors. By default, this package employs an alternative estimator due to Engle et al. (2019), which is better suited to large-dimensional problems. It achieves this by i) estimating $\bar{Q}$ with a nonlinear shrinkage estimator instead of the sample covariance of ϵ_t, and ii) estimating the DCC parameters by maximizing the sum of the pairwise log-likelihoods, rather than the joint log-likelihood over all assets, thereby avoiding the inversion of large matrices during the optimization. The estimation method is controlled by passing the `method` keyword to the constructor. Possible values are `:largescale` (the default), and `:twostep`.

Electronic copy available at: https://ssrn.com/abstract=3551503

CCC The CCC (constant conditional correlation) model of Bollerslev (1990) models $R_t = R$ as constant. It is the special case of the DCC model in which $p = q = 0$:

```
julia> CCC == DCC{0, 0}
true
```

As such, the constructor has the exact same signature, except that the DCC parameters must be passed as a zero-length vector:

```
julia> CCC([1. .5; .5 1.], Float64[], [GARCH{1, 1}([1., .9, .05]) for _ in 1:2])
DCC{0, 0, ARCHModels.TGARCH{0,1,1}} specification.

No estimable parameters.
```

As for the DCC model, the constructor accepts a `method` keyword argument, which takes the possible values `:largescale` (default) or `:twostep` that determines whether R will be estimated by nonlinear shrinkage or the sample correlation of the ϵ_t.

3.3.2 Mean Specifications

The conditional mean of a `MultivariateARCHModel` is specified by a vector of univariate `MeanSpec`s.

3.3.3 Multivariate Standardized Distributions

Multivariate standardized distributions subtype `MultivariateStandardizedDistribution`. Currently, only `MultivariateStdNormal` is available. Note that under mild assumptions, the Gaussian (quasi-)MLE consistently estimates the (multivariate) ARCH parameters even if Gaussianity is violated.

4 Illustrations

4.1 Univariate Modelling

We will be using the data from Bollerslev and Ghysels (1996), available as the constant `BG96`. The data consist of daily German mark/British pound exchange rates (1974 observations) and are often used in evaluating implementations of (G)ARCH models (see, e.g., Brooks et al. (2001). We begin by convincing ourselves that the data exhibit ARCH effects; a quick and dirty way of doing this is to look at the sample autocorrelation function of the squared returns:

```
julia> using ARCHModels

julia> autocor(BG96.^2, 1:4, demean=true)' # re-exported from StatsBase
1*4 LinearAlgebra.Adjoint{Float64,Array{Float64,1}}:
 0.222941  0.176632  0.14086  0.12632
```

Using a critical value of $1.96/\sqrt{1974} = 0.044$, we see that there is indeed significant autocorrelation in the squared series.

A more formal test for the presence of volatility clustering is Engle's (1982) ARCH-LM test. The test statistic is given by $LM \equiv TR^2_{aux}$, where R^2_{aux} is the coefficient of determination in a regression of the squared returns on an intercept and p of their own lags. The test statistic follows a χ^2_p distribution under the null of no volatility clustering.

```
julia> ARCHLMTest(BG96, 1)
ARCH LM test for conditional heteroskedasticity
------------------------------------------------
Population details:
    parameter of interest:   T·R^2 in auxiliary regression
    value under h_0:         0
    point estimate:          98.12107516935244
```


Electronic copy available at: https://ssrn.com/abstract=3551503

```
Test summary:
    outcome with 95% confidence: reject h_0
    p-value:                     <1e-22

Details:
    sample size:                   1974
    number of lags:                1
    LM statistic:                  98.12107516935244
```

The null is strongly rejected, again providing evidence for the presence of volatility clustering.

Having established the presence of volatility clustering, we can begin by fitting the workhorse model of volatility modeling, a GARCH(1, 1) with standard normal errors; for other model classes such as EGARCH, see Section 3.1.1.

```
julia> fit(GARCH{1, 1}, BG96)

ARCHModels.TGARCH{0,1,1} model with Gaussian errors, T=1974.

Mean equation parameters:
```

	Estimate	Std.Error	z value	Pr(>|z|)
μ	-0.00616637	0.00920163	-0.670139	0.5028

Volatility parameters:

	Estimate	Std.Error	z value	Pr(>|z|)
ω	0.0107606	0.00649493	1.65677	0.0976
β_1	0.805875	0.0725003	11.1155	<1e-27
α_1	0.153411	0.0536586	2.85903	0.0042

This returns an instance of `UnivariateARCHModel`, as described in Section 3.2. The parameters α_1 and β_1 in the volatility equation are highly significant, as one would expect from our preliminary testing. Note also that the fitted values are the same as those found by Bollerslev and Ghysels (1996) and Brooks et al. (2001) for the same dataset.

The `fit` method supports a number of keyword arguments; the full signature is

```
fit(
    ::Type{<:UnivariateVolatilitySpec},
    data::Vector;
    dist=StdNormal,
    meanspec=Intercept,
    algorithm=BFGS(),
    autodiff=:forward,
    kwargs...
    )
```

Here, `dist` is a subtype (not instance) of `StandardizedDistribution`; see Section 3.1.3. The mean specification is specified via `meanspec` and defaults to `Intercept`. It can be passed as either a subtype of `MeanSpec` or an instance thereof (for specifications that require additional data, such as `Regression`; see Section 3.1.2). If the mean specification in question has a notion of sample size (like `Regression`), then the sample size should match that of the data, or an error will be thrown. The remaining keyword arguments are passed on to the optimizer.

Electronic copy available at: https://ssrn.com/abstract=3551503

As an example, an EGARCH(1, 1, 1) model with an intercept and Student's t errors would be fitted as follows:

```
julia> fit(EGARCH{1, 1, 1}, BG96; meanspec=Intercept, dist=StdT);
```

An alternative approach to fitting a `UnivariateVolatilitySpec` to `BG96` is to first construct a `UnivariateARCHModel` containing the data, and then using `fit!` to modify it in place:

```
julia> am = UnivariateARCHModel(GARCH{1, 1}([1., 0., 0.]), BG96);

julia> fit!(am);
```

Calling `fit(am)` will return a new instance of `UnivariateARCHModel` instead.

It is also possible to pass a `LinearModel` (or `TableRegressionModel`) from **GLM.jl** to `fit` instead of a data vector. This is equivalent to using a `Regression` as a mean specification. In the following example, we fit a linear model with GARCH(1, 1) errors, where the design matrix consists of a breaking intercept and time trend:

```
julia> using GLM, DataFrames

julia> data = DataFrame(B=[ones(1000); zeros(974)], T=1:1974, Y=BG96);

julia> model = lm(@formula(Y ~ B*T), data);

julia> fit(GARCH{1, 1}, model);
```

One of the issues in ARCH modeling is selecting the lag order. One possibility is to make this choice based on an information criterion. **ARCHModels.jl** can automate this procedure, via the `selectmodel` function. Given a class of model (i.e., a subtype of `UnivariateVolatilitySpec`), it will return a fitted `UnivariateARCHModel`, with the lag length parameters (i.e., p and q in the case of GARCH) chosen to minimize the desired criterion. The BIC is used by default. As an example, the following selects the optimal (minimum AIC) EGARCH(o, p, q) model, where $o, p, q < 2$, assuming t distributed errors.

```
julia> selectmodel(EGARCH, BG96; criterion=aic, maxlags=2, dist=StdT)

ARCHModels.EGARCH{1,1,2} model with Student's t errors, T=1974.

Mean equation parameters:
```

	Estimate	Std.Error	z value	Pr(>|z|)
μ	0.00196126	0.00695292	0.282077	0.7779

```
Volatility parameters:
```

	Estimate	Std.Error	z value	Pr(>|z|)
ω	-0.0031274	0.0112456	-0.278101	0.7809
γ_1	-0.0307681	0.0160754	-1.91398	0.0556
β_1	0.989056	0.0073654	134.284	<1e-99
α_1	0.421644	0.0678139	6.21767	<1e-9
α_2	-0.229068	0.0755326	-3.0327	0.0024

```
Distribution parameters:
```

Estimate	Std.Error	z value	Pr(>|z|)

Electronic copy available at: https://ssrn.com/abstract=3551503

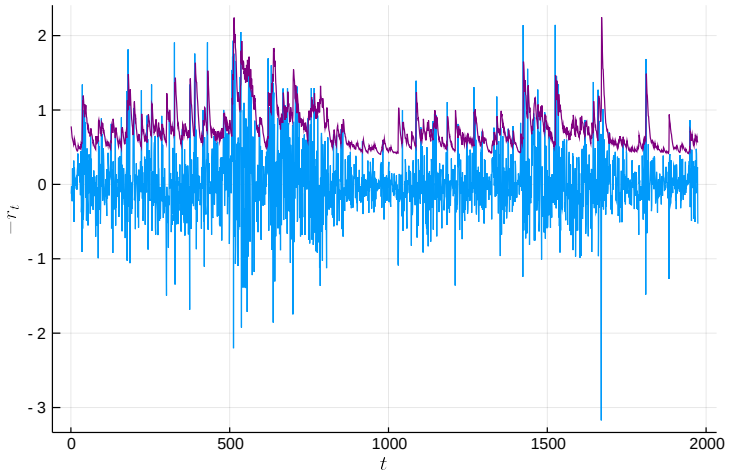

Table 1: 5% Value at Risk for the Bollerslev and Ghysels data.

ν	4.18795	0.418697	10.0023	<1e-22

Here, an EGARCH(1, 1, 2) model was selected. Passing the keyword argument `show_trace=true` will show the criterion for each model after it is estimated. Any unspecified lag length parameters in the mean specification (e.g., p and q for ARMA) will be optimized over as well. Note however that this can result in an explosion of the number of models that must be estimated; e.g., selecting the best model from the class of TGARCH(o, p, q)-ARMA(p, q) models results in $5^{\texttt{maxlags}}$ models being estimated. It may thus be preferable to fix the lag length of the mean specification: `am = selectmodel(ARCH, BG96; meanspec=AR{1})` considers only ARCH(q)-AR(1) models. Similarly, one may restrict the lag length of the volatility specification and select only among different mean specifications. E.g., the following will select the best ARMA(p, q) specification with constant variance:

```
julia> am = selectmodel(ARCH{0}, BG96;  meanspec=ARMA);
```

One of the primary uses of ARCH models is for estimating and forecasting Value at Risk. Basic in-sample estimates for the Value at Risk implied by an estimated `UnivariateARCHModel` can be obtained using `VaRs`:

```
julia> am = fit(GARCH{1, 1}, BG96);

julia> vars = VaRs(am, 0.05);

julia> using Plots

julia> plot(-BG96, legend=:none, xlabel="\$t\$", ylabel="\$-r_t\$");

julia> plot!(vars, color=:purple);

julia> savefig(joinpath("img", "VaRplot.pdf")); nothing
```

This produces the graph in Figure 1.

The `predict(am::UnivariateARCHModel)` method can be used to construct one-step ahead forecasts for a number of quantities. Its signature is

```
predict(am::UnivariateARCHModel, what=:volatility; level=0.01)
```

The keyword argument `what` controls which object is predicted; the choices are `:volatility` (the default), `:variance`, `:return`, and `:VaR`. The VaR level can be controlled with the keyword argument `level`.

Electronic copy available at: https://ssrn.com/abstract=3551503

One way to use `predict` is in a backtesting exercise. The following code snippet constructs out-of-sample VaR forecasts for the Bollerslev and Ghysels data by re-estimating the model in a rolling window fashion, and then tests the correctness of the VaR specification with the dynamic quantile test of Engle and Manganelli (2004) via `DQTest`.

```
julia> T = length(BG96);

julia> windowsize = 1000;

julia> vars = similar(BG96);

julia> for t = windowsize+1:T-1
    m = fit(GARCH{1, 1}, BG96[t-windowsize:t]);
    vars[t+1] = predict(m, :VaR; level=0.05);
end

julia> DQTest(BG96[windowsize+1:end], vars[windowsize+1:end], 0.05)
Engle and Manganelli's (2004) DQ test (out of sample)
-----------------------------------------------------
Population details:
    parameter of interest:   Wald statistic in auxiliary regression
    value under h_0:         0
    point estimate:          1.27596894825603

Test summary:
    outcome with 95% confidence: fail to reject h_0
    p-value:                     0.7348

Details:
    sample size:                   974
    number of lags:                1
    VaR level:                     0.05
    DQ statistic:                  1.27596894825603
```

Testing volatility models in general relies on the estimated conditional volatilities $\hat{\sigma}_t$ and the standardized residuals $\hat{z}_t \equiv (r_t - \hat{\mu}_t)/\hat{\sigma}_t$, accessible via `volatilities(::UnivariateARCHModel)` and `residuals(::UnivariateARCHModel)`, respectively. The non-standardized residuals $\hat{u}_t \equiv r_t - \hat{\mu}_t$ can be obtained by passing `standardized=false` as a keyword argument to `residuals`.

One possibility to test a volatility specification is to apply the ARCH-LM test to the standardized residuals. This is achieved by calling `ARCHLMTest` on the estimated `UnivariateARCHModel`:

```
julia> am = fit(GARCH{1, 1}, BG96);

julia> ARCHLMTest(am, 4) # 4 lags in test regression.
ARCH LM test for conditional heteroskedasticity
-----------------------------------------------
Population details:
    parameter of interest:   T·R² in auxiliary regression
    value under h_0:         0
    point estimate:          4.211230445141555

Test summary:
    outcome with 95% confidence: fail to reject h_0
    p-value:                     0.3782

Details:
    sample size:                   1974
```


Electronic copy available at: https://ssrn.com/abstract=3551503

```
number of lags:                 4
LM statistic:                   4.211230445141555
```

By default, the number of lags is chosen as the maximum order of the volatility specification (e.g., $\max(p, q)$ for a GARCH(p, q) model). Here, the test does not reject, indicating that a GARCH(1, 1) specification is sufficient for modelling the volatility clustering (a common finding).

To simulate from a `UnivariateARCHModel`, the `simulate` function is used. One can either specify the `UnivariateVolatilitySpec` (and optionally the distribution and mean specification) and desired number of observations, or pass an existing `UnivariateARCHModel`. Using `simulate!` modifies the data in place instead. Example:

```
julia> am3 = simulate(
    GARCH{1, 1}([1., .9, .05]),
    1000;
    warmup=500,
    meanspec=Intercept(5.),
    dist=StdT(3.)
    );

julia> am4 = simulate(am3, 1000);
```

Care must be taken if the mean specification has a notion of sample size, as in the case of `Regression`: because the sample size must match that of the data to be simulated, one must pass `warmup=0`, or an error will be thrown. For example, `am3` above could also have been simulated from as follows:

```
julia> reg = Regression([5], ones(1000, 1));

julia> am3 = simulate(
    GARCH{1, 1}([1., .9, .05]),
    1000;
    warmup=0,
    meanspec=reg,
    dist=StdT(3.)
    );
```

4.2 Multivariate models

In this section, the percentage returns on 29 stocks from the DJIA from 03/19/2008 through 04/11/2019, available as `DOW29`, will be used.

Fitting a multivariate ARCH model proceeds similarly to the univariate case, by passing the type of the multivariate ARCH specification to `fit`. If the lag length (and in the case of the DCC model, the univariate specification) is left unspecified, then these default to 1 (and GARCH); i.e., the following is equivalent to both `fit(DCC{1, 1}, DOW29)` and `fit(DCC{1, 1, GARCH{1, 1}}, DOW29)`:

```
julia> m = fit(DCC, DOW29[:, 1:2]);
```

The returned object is of type `MultivariateARCHModel`. Like `UnivariateARCHModel`, it implements most of the interface of `StatisticalModel` and hence behaves similarly, so this section documents only the major differences.

The standard errors are not calculated by default. They can be obtained with `show(IOContext(stdout, :se=>true), m)`. Alternatively, `stderror(m)` can be used. As in the univariate case, `fit` supports a number of keyword arguments. The full signature is

```
fit(spec, data: method=:largescale,  dist=MultivariateStdNormal, meanspec=Intercept,
      algorithm=BFGS(), autodiff=:forward, kwargs...)
```

Their meaning is similar to the univariate case. In particular, `meanspec` can be any univariate mean specification, as described in Section 3.1.2. Certain models support different estimation methods; in the case of the DCC model, these are `:twostep` and `:largescale`, which respectively refer to the methods

Electronic copy available at: https://ssrn.com/abstract=3551503

of Engle (1982) and Engle et al. (2019). The latter sacrifices some amount of statistical efficiency for much-improved computational speed and is the default. Again paralleling the univariate case, one may also construct a `MultivariateARCHModel` by hand and then call `fit` or `fit!` on it, but this is rather cumbersome, as it requires specifying all parameters of the covariance specification. One-step ahead forecasts of the covariance or correlation matrix are obtained by respectively passing `what=:covariance` (the default) or `what=:correlation` to `predict`:

```
julia> predict(m, what=:correlation)
2*2 Array{Float64,2}:
 1.0       0.436513
 0.436513  1.0
```

In the multivariate case, there are three types of residuals that can be considered: the unstandardized residuals, a_t; the devolatized residuals, ϵ_t, where $\epsilon_{it} \equiv a_{it}/\sigma_{it}$; and the decorrelated residuals $z_t \equiv \Sigma_t^{-1/2} a_t$. When called on a `MultivariateARCHModel`, `residuals` returns $\{z_t\}$ by default. Passing `decorrelated=false` returns $\{\epsilon_t\}$, and passing `standardized=false` returns $\{a_t\}$ (`decorrelated=true` implies `standardized=true`).

5 Comparison with other packages

5.1 Numerical results

Brooks et al. (2001) conduct a comparison of different implementations of the GARCH(1, 1) model. As a benchmark, they use the estimates reported by Bollerslev and Ghysels (1996) for a German mark/British pound data set, available as `BG96` in **ARCHModels.jl**. The estimates reported by Bollerslev and Ghysels (1996) are $\mu = -0.00619$ (-0.67), $\omega = 0.0108$ (1.66), $\beta_1 = 0.806$ (11.11), $\alpha_1 = 0.153$ (2.86); values in parentheses represent standard errors. Here, we test Julia's **ARCHModels.jl**, Matlab's **Econometrics Toolbox**, Python's **ARCH** package, and R's **rugarch** package on the same data set.

Julia

```
julia> using ARCHModels

julia> fit(GARCH{1, 1}, BG96, meanspec=Intercept)

ARCHModels.TGARCH{0,1,1} model with Gaussian errors, T=1974.

Mean equation parameters:
```

	Estimate	Std.Error	z value	Pr(>|z|)
μ	-0.00616637	0.00920163	-0.670139	0.5028

```
Volatility parameters:
```

	Estimate	Std.Error	z value	Pr(>|z|)
ω	0.0107606	0.00649493	1.65677	0.0976
β_1	0.805875	0.0725003	11.1155	<1e-27
α_1	0.153411	0.0536586	2.85903	0.0042

Matlab Note: LaTeX package **PythonTex** does not support Matlab yet, so we call it from Julia.

```
julia> using MATLAB
```


Electronic copy available at: https://ssrn.com/abstract=3551503

```
julia> mat"version"
"9.7.0.1190202 (R2019b)"

julia> mat"estimate(garch('ARCHLags', 1, 'GARCHLags', 1, 'Offset', NaN), $BG96); 0;"

    GARCH(1,1) Conditional Variance Model with Offset (Gaussian Distribution):

                  Value       StandardError    TStatistic      PValue
                 ----------    -------------    ----------    ----------

    Constant      0.010761       0.001323         8.1342     4.1453e-16
    GARCH{1}       0.80597        0.01656         48.669              0
    ARCH{1}        0.15313       0.013974         10.959     6.0379e-28
    Offset      -0.0061904      0.0084336       -0.73402        0.46294

0.0
```

Python

```
>>> import pandas as pd
>>> import timeit
>>> import functools
>>> from arch.univariate.mean import ConstantMean
>>> from arch.univariate.distribution import Normal
>>> from arch.univariate import GARCH
>>> data = pd.read_csv("http://people.stern.nyu.edu/wgreene/"
...     "Text/Edition7/TableF20-1.txt").to_numpy().flatten()
>>> am = ConstantMean(data, rescale=False)
>>> am.volatility = GARCH(1, 0, 1)
>>> am.fit(disp='off')
                    Constant Mean - GARCH Model Results
==============================================================================
Dep. Variable:                          y   R-squared:                      -0.000
Mean Model:             Constant Mean   Adj. R-squared:                 -0.000
Vol Model:                      GARCH   Log-Likelihood:                -1104.52
Distribution:                  Normal   AIC:                            2217.04
Method:            Maximum Likelihood   BIC:                            2239.39
                                        No. Observations:                 1974
Date:                Mon, Mar 09 2020   Df Residuals:                     1970
Time:                        19:00:21   Df Model:                            4
                                 Mean Model
==============================================================================
                 coef    std err          t      P>|t|       95.0% Conf. Int.
------------------------------------------------------------------------------
mu         -6.0765e-03  9.197e-03     -0.661      0.509 [-2.410e-02,1.195e-02]
                              Volatility Model
==============================================================================
                 coef    std err          t      P>|t|       95.0% Conf. Int.
------------------------------------------------------------------------------
omega      9.9151e-03  6.341e-03      1.564      0.118 [-2.512e-03,2.234e-02]
alpha[1]       0.1455  5.573e-02      2.611  9.039e-03    [3.626e-02,  0.255]
beta[1]        0.8168  7.445e-02     10.971  5.253e-28      [  0.671,  0.963]
==============================================================================

Covariance estimator: robust
ARCHModelResult, id: 0x221e05e8748
```


Electronic copy available at: https://ssrn.com/abstract=3551503

R

```
> suppressPackageStartupMessages(library("rugarch"))
Warning message:
package 'rugarch' was built under R version 3.6.3
> data(dmbp)
> spec = ugarchspec(variance.model=list(model="sGARCH", garchOrder=c(1, 1)),
+     mean.model=list(armaOrder=c(0, 0), include.mean=TRUE))
> ugarchfit(data=dmbp, spec=spec)

*---------------------------------*
*          GARCH Model Fit        *
*---------------------------------*

Conditional Variance Dynamics
-----------------------------------
GARCH Model    : sGARCH(1,1)
Mean Model     : ARFIMA(0,0,0)
Distribution   : norm

Optimal Parameters
------------------------------------
        Estimate  Std. Error  t value Pr(>|t|)
mu     -0.006185    0.008462 -0.73094 0.464813
omega   0.010760    0.002853  3.77155 0.000162
alpha1  0.153407    0.026581  5.77124 0.000000
beta1   0.805880    0.033567 24.00821 0.000000

Robust Standard Errors:
        Estimate  Std. Error  t value Pr(>|t|)
mu     -0.006185    0.009017 -0.68594 0.492752
omega   0.010760    0.006498  1.65582 0.097758
alpha1  0.153407    0.049390  3.10606 0.001896
beta1   0.805880    0.069163 11.65196 0.000000

LogLikelihood : -1106.587

Information Criteria
------------------------------------

Akaike       1.1252
Bayes        1.1365
Shibata      1.1252
Hannan-Quinn 1.1294

Weighted Ljung-Box Test on Standardized Residuals
------------------------------------
                        statistic p-value
Lag[1]                      5.061 0.02447
Lag[2*(p+q)+(p+q)-1][2]     5.167 0.03671
Lag[4*(p+q)+(p+q)-1][5]     6.468 0.06964
d.o.f=0
H0 : No serial correlation

Weighted Ljung-Box Test on Standardized Squared Residuals
------------------------------------
```


Electronic copy available at: https://ssrn.com/abstract=3551503

```
                        statistic p-value
Lag[1]                      2.501  0.1137
Lag[2*(p+q)+(p+q)-1][5]     3.575  0.3120
Lag[4*(p+q)+(p+q)-1][9]     5.474  0.3632
d.o.f=2

Weighted ARCH LM Tests
------------------------------------
            Statistic Shape Scale P-Value
ARCH Lag[3]     1.668 0.500 2.000  0.1965
ARCH Lag[5]     1.673 1.440 1.667  0.5481
ARCH Lag[7]     3.603 2.315 1.543  0.4075

Nyblom stability test
------------------------------------
Joint Statistic:  0.6641
Individual Statistics:
mu     0.1673
omega  0.3550
alpha1 0.2635
beta1  0.3349

Asymptotic Critical Values (10% 5% 1%)
Joint Statistic:          1.07 1.24 1.6
Individual Statistic:     0.35 0.47 0.75

Sign Bias Test
------------------------------------
                   t-value   prob sig
Sign Bias           1.3192 0.1873
Negative Sign Bias  0.2434 0.8077
Positive Sign Bias  0.6660 0.5055
Joint Effect        2.8773 0.4109


Adjusted Pearson Goodness-of-Fit Test:
------------------------------------
  group statistic p-value(g-1)
1    20     109.9    8.208e-15
2    30     137.9    4.003e-16
3    40     137.4    6.866e-13
4    50     160.5    8.747e-14


Elapsed time : 0.1004241
```

We find the parameter estimates obtained by all packages to be quite similar, but the standard errors reported by Matlab are wildly off.

5.2 Benchmarks

Next, we compare the different implementations in terms of speed.

Julia

```
julia> using BenchmarkTools
```


Electronic copy available at: https://ssrn.com/abstract=3551503

```
julia> @btime m =  fit(GARCH{1, 1}, $BG96, meanspec=Intercept);
  3.050 ms (2253 allocations: 421.33 KiB)
```

Matlab

```
julia> mat"mdl = garch('ARCHLags', 1, 'GARCHLags', 1, 'Offset', NaN);"

julia> mat"f = @() estimate(mdl, $BG96, 'Display', 'off'); timeit(f)"
0.0376169377
```

Python

```
>>> N = 100
>>> timeit.timeit(functools.partial(am.fit, disp="off"), number=N)/N
0.020592828
```

R

```
> library("microbenchmark")
Warning message:
package 'microbenchmark' was built under R version 3.6.3
> microbenchmark(ugarchfit(data=dmbp, spec=spec), times=10)
Unit: milliseconds
                              expr      min       lq     mean   median
 ugarchfit(data = dmbp, spec = spec) 135.1914 152.0893 161.8299 156.3239
       uq     max neval
 159.3782 238.915    10
```

We find that **ARCHModels.jl** is faster than Python's **ARCH** package by a factor of about 6, faster than Matlab's **Econometrics Toolbox** by a factor of about 12, and faster than R's **rugarch** package by a factor of about 50. This is despite the fact that all packages (except **ARCHModels.jl**) implement the likelihood function in a compiled low-level language.

6 Summary

We have introduced **ARCHModels.jl**, a package for estimating ARCH-type models in Julia. It provides an intuitive API for estimating, simulating, and testing univariate and multivariate ARCH-type models. Julia's properties, such as JIT compilation and the availability of advanced automatic differentiation packages, allow us to beat implementations in other languages by significant margins in terms of estimation speed, despite keeping the entire implementation in the high-level language.

Acknowledgments

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 750559).

References

Bates, D. and other contributors (2012). **JuliaStats/GLM.jl**. 1

Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98. 1

Bollerslev, T. (1986). Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31:307–327. 1, 2

Electronic copy available at: https://ssrn.com/abstract=3551503

Bollerslev, T. (1990). Modelling the coherence in short-run nominal exchange rates: A multivariate generalized arch model. *The Review of Economics and Statistics*, 72(3):498–505. 6

Bollerslev, T. and Ghysels, E. (1996). Periodic autoregressive conditional heteroscedasticity. *Journal of Business & Economic Statistics*, 14:139–151. 6, 7, 9, 10, 12

Brooks, C., Burke, S., and Persand, G. (2001). Benchmarks and the accuracy of garch model estimation. *International Journal of Forecasting*, 17:45–56. 6, 7, 12

Engle, R. (2002). Dynamic conditional correlation. *Journal of Business & Economic Statistics*, 20(3):339–350. 5

Engle, R. F. (1982). Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica*, 50(4):987–1007. 1, 2, 6, 12

Engle, R. F., Ledoit, O., and Wolf, M. (2019). Large dynamic covariance matrices. *Journal of Business & Economic Statistics*, 37(2):363–375. 5, 12

Engle, R. F. and Manganelli, S. (2004). Caviar. *Journal of Business & Economic Statistics*, 22(4):367–381. 10

Ghalanos, A. (2019). **rugarch***: Univariate GARCH models.* R package version 1.4-1. 1

Glosten, L. R., Jagannathan, R., and Runkle, D. E. (1993). On the relation between the expected value and the volatility of the nominal excess return on stocks. *The Journal of Finance*, 48:1779–1801. 2

Harris, H., DuBois, C., White, J. M., and other contributors (2015). **JuliaData/DataFrames.jl**. 1

Kornblith, S. and other contributors (2012). **JuliaStats/HypothesisTests.jl**. 1

Lin, D., Byrne, S., Noack, A., Bates, D., White, J. M., Kornblith, S., and other contributors (2019a). **JuliaStats/StatsBase.jl**. 1

Lin, D., White, J. M., Byrne, S., Bates, D., Noack, A., Pearson, J., Arslan, A., Squire, K., Anthoff, D., Papamarkou, T., Besançon, M., Drugowitsch, J., Schauer, M., and other contributors (2019b). **JuliaStats/Distributions.jl**: a Julia package for probability distributions and associated functions. 1

Mogensen, P. K. and Riseth, A. N. (2018). Optim: A mathematical optimization package for Julia. *Journal of Open Source Software*, 3(24):615. 1

Nelson, D. B. (1991). Conditional heteroskedasticity in asset returns: A new approach. *Econometrica*, 59:347–370. 2

R Core Team (2019). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. 1

Revels, J., Lubin, M., and Papamarkou, T. (2016). Forward-mode automatic differentiation in julia. *arXiv:1607.07892 [cs.MS]*. 1

Sheppard, K. and other contributors (2019). **ARCH**. 1

The MathWorks Inc. (2019). Matlab – the language of technical computing. Version 2019b. 1

van Rossum, G. (1995). Python tutorial. Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam. 1

Electronic copy available at: https://ssrn.com/abstract=3551503